兰州大学

本科毕业论文

论文题目（中文） 深度 Koopman 算子辅助的非线性系统强化学习研究

论文题目（英文） Deep Koopman Operator Assisted Reinforcement Learning of Nonlinear System

学生姓名 许恣欢

指导教师 赵东东

学　　院 信息科学与工程学院

专　　业 电子信息科学与工程

年　　级 2020 级

兰州大学教务处

诚信责任书

本人郑重声明：本人所呈交的毕业论文（设计），是在导师的指导下独立进行研究所取得的成果。毕业论文（设计）中凡引用他人已经发表或未发表的成果、数据、观点等，均已明确注明出处。除文中已经注明引用的内容外，不包含任何其他个人、集体已经发表或未发表的论文。

本声明的法律责任由本人承担。

论文作者签名：许悉欢　　日　　期：2024年5月28日

关于毕业论文（设计）使用授权的声明

本人在导师指导下所完成的论文及相关的职务作品，知识产权归属兰州大学。本人完全了解兰州大学有关保存、使用毕业论文（设计）的规定，同意学校保存或向国家有关部门或机构送交论文的纸质版和电子版，允许论文被查阅和借阅；本人授权兰州大学可以将本毕业论文（设计）的全部或部分内容编入有关数据库进行检索，可以采用任何复制手段保存和汇编本毕业论文（设计）。本人离校后发表、使用毕业论文（设计）或与该毕业论文（设计）直接相关的学术论文或成果时，第一署名单位仍然为兰州大学。

本毕业论文（设计）研究内容：

☑可以公开

□不宜公开，已在学位办公室办理保密申请，解密后适用本授权书。

（请在以上选项内选择其中一项打“√”）

论文作者签名：许悉欢　　导师签名：赵东东

日　　期：2024年5月28日　　日　　期：2024年5月28日

深度 Koopman 算子辅助的非线性系统强化学习研究

中文摘要

在这篇文章中，提出了通过改进强化学习框架来更好地控制非线性系统的一种算法。首先，引入了熵正则策略优化，以加强策略的探索，平衡数据的探索和利用，以避免过早收敛的问题，并最大化回报。然后，为了解决非线性系统复杂动力学的挑战，引入了 Koopman 算子理论，并强调了如何通过数据驱动的方式得到 Koopman 算子及其观测函数，利用 Koopman 嵌入函数将系统动力学嵌入到更高维的线性空间中，从而实现线性地推进系统动力学。最后，分两步提出了深度 Koopman 算子辅助的最大熵强化学习算法，具体地，通过改写贝尔曼方程并将价值估计提升到嵌入空间，实现了在高维线性空间上线性推进价值函数，从而提升了训练效果。实验证明，在具有连续谱的非线性 Lorenz 系统中，我们的算法在 50000 步的训练和探索中表现优于普通的深度强化学习和线性控制算法。

关键词： Koopman 算子理论，深度神经网络，强化学习

Deep Koopman Operator Assisted Reinforcement Learning of Nonlinear System

Abstract

In this article, I propose an algorithm to control nonlinear systems more precisely by improving the reinforcement learning framework. First, entropy regular policy optimization is added to enhance the exploration of strategies, balance the exploration and utilization of data to avoid premature convergence problems, finally maximize returns. Then, in order to solve the challenges of complex dynamics of nonlinear systems, the Koopman operator theory is introduced, and this paper emphasizes how to obtain the Koopman operator and its observation function in a data-driven manner. Using Koopman embedding function to embed system dynamics into higher dimensional linear space, so as to achieve linear propulsion of system dynamics. Finally, a deep Koopman operator-assisted maximum entropy reinforcement learning algorithm is proposed in two steps. Specifically, by rewriting the Bellman equation and lifting the value estimate to the embedding space, the value function is achieved linearly in the high-dimensional linear space, thus improving the training effect. Experiments show that in a nonlinear Lorenz system with continuous spectrum, our algorithm outperforms ordinary deep reinforcement learning and linear control algorithms in training and exploration at 50,000 steps.

Keywords: Koopman Operator Theory, Deep Neural Network, Reinforcement Learning

目　录

第一章 绪 论

1.1 研究背景及意义

复杂非线性系统的分析和控制一直以来都是国内外的研究热点。如今，线性系统的控制已经形成了一套比较完善的体系，传统的控制手段如 PID、LQR 等已经在实际应用中得到了验证。但是非线性系统由于其难以描述的系统动力学，线性系统方案在精确研究其变化上存在困难。目前应对这种困难比较多的方法是将非线性系统线性化，通过线性控制理论对系统进行控制。现行的做法有两种，第一种是通过局部化研究范围，使用近似法对系统进行线性化，这样的做法只能实现局部线性化，并不能完善地表示整个系统空间内的动态，在远离研究点邻域时效果很差。第二种则是使用全局线性化理论，例如 Koopman 算子理论，这个方法线性化的主要方式是使用一个系统变量的观测函数，假设其为无限维的，那么在这样一个坐标系下，系统变量的演化可被认为是线性的，Koopman 理论还提出，在无限维的观测空间下，可以使用 Koopman 算子推进系统线性演化，值得注意的是，在无限维的情况下，这样的做法是不损耗系统信息的。

然而，无限维的 Koopman 算子和观测函数是难以得到的，所以很多工作都着重于得到 Koopman 算子的有限维近似，如扩展动态模式分解（EDMD。此外，使用 Koopman 算子理论得到系统信息是一次性的工作，使用数据驱动方式得到 Koopman 算子的方式也流行起来。使用深度学习方式使用大量数据得到 Koopman 算子，并用额外的网络得到 Koopman 算子矩阵特征值的方式 [1]，在实现比较好的效果的情况下，保留了系统的可解释性。利用数据驱动的 Koopman 模型设计控制器的一些早期工作设计使用 DMD 和 eDMD 模型进行模型预测控制 [2]，最近的研究，如增量更新 [3] 和 Koopman 自动编码器 [4]。重要的是，这些方法没有能够得到收敛保证，为解决这些问题，出现了以强化学习方式进行策略优化的 Koopman 控制算法研究 [5]。

强化学习在控制领域的研究在前些年取得了突破性的发展，最早提出的 Q-learning 算法着重考察在离散动作空间中，能够得到最大回报的动作，将研究限制在了拥有有限个动作的系统内；Lillicrap 等人 [6] 在 Actor-Critic 框架的基础上，使用神经网络来参数化策略函数，提出了深度确定性策略梯度（DDPG）算法，使用深度确定性策略网络，使得策略能够在连续动作空间中，做出确定性决策；Haarnoja 等人 [7] 提出了最大熵强化学习（SAC），在策略优化目标中加入熵正则项，并令策略项为能量的指数项，在解决连续空间的基础上，加强了对状态空间完整的探索。而 Koopman 算子的加入能够为强化学习提供策略优化和价值判断的指向性，对于拥有高维状态和非线性系统来说，这样的方法是有效的。

然而，对于复杂多变的非线性系统而言，往往会发现 Koopman 算子是无限维的，继续

使用基于模型的算法将会举步维艰。近年来深度神经网络发展迅速，如多层感知机（MLP），其对于未知函数的参数化的能力很强，具有潜力实现 Koopman 算子的数据驱动框架，并可以用于发现和表示 Koopman 特征函数，提供了对强非线性系统进行升维并线性表示的可能性。DNN 表示的 Koopman 特征函数，即使对于高维和强非线性系统也保持可解释和简洁。深度学习方法在 Koopman 算子理论上的应用，具有很强的适应性和扩展性，同时在经典动力学系统方面保持直观性。

1.2 国内外研究现状

1.2.1 深度强化学习

强化学习是一个快速发展的领域，处于机器学习和控制理论的交叉点，其中智能体学习在复杂的环境进行交互以实现目标 [8]。

强化学习分为两类，一种是基于价值的强化学习，其优化目标为使得价值函数对于动作空间的评估更加精确，在探索中，通过选取最优动作来获得最大奖励；另一种是基于策略的强化学习，通过参数化，使用策略梯度的方式，最优化策略，以达到更好地控制。Watkins [9] 提出的 Q-learning 算法就是最经典的基于价值的强化学习算法，在离散的状态和动作空间中，学习价值表，并通过选取最优价值动作最大化奖励。此后，为了解决在连续空间中的控制，Mnih 等人提出了 DQN 算法 [10]，在 DQN 算法中，使用卷积神经网络来获取连续空间数据中的特征，再通过价值网络进行价值评估并优化，强化学习进入深度时代；在此基础上，又提出了学习特定策略而不是最优策略的 SARSA 算法，此算法将在以后的 Actor-Critic 框架中发挥巨大作用；Sutton [11] 提出了策略梯度（Policy Gradient）的计算方式，基于策略的强化学习得以实现，策略的训练目标变为优化策略网络，以使算法收敛；而将价值优化和策略优化一起进行的算法框架叫做 Actor-Critic 框架，训练时，策略网络从价值网络的输出中获取信息，并通过梯度上升的方式，优化自身参数，而价值网络通过策略探索，使用普通的价值优化的方式使得估计值更真，如今基于策略的现代强化学习算法大多基于此框架；Lillicrap 等人 [6] 提出的深度确定性策略梯度算法采用神经网络拟合连续动作空间中的确定性策略，实现了连续状态和空间系统的强化学习优化控制变得可能；不久，Haarnoja 等人 [7] 便提出了 Soft Actor-Critic 算法，提高了对动作空间的好奇性，平衡了对样本的利用和对系统的探索；

深度强化学习最近被证明能够在几个具有挑战性的任务 [12, 13] 中实现人类水平或超人类的表现，包括玩视频游戏 [10, 14] 和策略游戏 [15–17]。深度强化学习也越来越多地用于科学和工程应用，包括用于药物发现 [18]、机器人操作 [19]、自动驾驶 [20] 和无人机竞速 [21]、流体流动控制 [22–26] 和聚变控制 [27]。

尽管取得了上述进展，深度强化学习解决方案通常需要巨大的计算资源来训练和泛化，并且它们通常缺乏在许多应用中至关重要的可解释性。许多强化学习策略背后的算法与优

化和最优控制理论 [8] 中的贝尔曼方程（Bellman Equation）及其连续形式汉密尔顿-雅可比-贝尔曼（Hamilton-Jacob-Bellman）方程密切相关。然而，求解这些方程对于高维非线性系统来说变得棘手，这通常可以使用深度学习来缓解。在 [5] 的工作中，通过 Koopman 算子的新应用改写贝尔曼方程，在无限维函数空间上将非线性动态系统重塑为线性方程组。

1.2.2 Koopman 算子理论

Koopman 算子理论最早由 B. O. Koopman 提出。其最大的优势是将非线性系统的动态用一组基底在无限维维空间中表示，而如何对这个空间进行最优近似，并得到一个有限维空间成了研究的重点。采用比较广泛的有基于数据驱动的动态模式分解方法，通过寻找 Koopman 的特征值和模态，来构建嵌入空间；后来的研究通过研究 Koopman 模态和 DMD 方法，进一步提出了拓展动态模式分解，即 eDMD 方式，来计算受控系统的 Koopman 算子；Bethany 等人 [1] 利用深度神经网络拟合嵌入函数和 Koopman 本征函数，并通过额外的神经网络表示 Koopman 特征函数，即使对于高维和强非线性系统也保持可解释和简洁性。

而在实际应用中，一般将非线性动力学映射到嵌入空间中的线性系统，而嵌入式线性系统常通过 LQR [28] 和 MPC [29] 等线性方法进行控制。尽管如此，选择嵌入函数以保持预测质量仍然是一项艰巨的任务。最近的方法侧重于用深度神经网络 [30] 学习嵌入函数，然后应用线性控制方法 [31]。在 [32] 中，提出了一个端到端的深度学习框架，将 Koopman 嵌入函数和 Koopman 算子一起学习，缓解了上述困难。

1.3 本文主要工作

本文主要研究了如何使用改进的最大熵强化学习算法实现对于非线性动力学系统的最优控制。本文利用了最大熵强化学习算法构建了非线性系统最优学习问题，并通过引入深度 Koopman 算子，对问题中涉及的强化学习对于强非线性的适应性问题进行改善，最后利用实现的深度 Koopman 算子辅助的最大熵强化学习算法对强非线性系统进行最优控制。

第一章为绪论，主要分析了使用深度 Koopman 算子和最大熵强化学习的研究背景和意义，并介绍了本文的结构。

第二章为基础知识，首先介绍了 Koopman 算子理论的基本概念，并阐述了高阶升维系统的得到和与原系统的数学关系；然后介绍本文使用的基础强化学习框架，并着重介绍了本文强化学习和其他算法的核心结合点：马尔可夫决策过程和贝尔曼方程；最后提及简单的神经网络架构，为本文中特殊函数参数化做铺垫。

第三章介绍了如何通过数据驱动的方式的到受控的、动作相关的 Koopman 算子，并介绍如何将其引入马尔可夫决策过程，进而改进价值估计，使得强化学习方式更好地解决非线性系统控制问题。

第四章介绍如何通过深度学习方式得到非线性动作相关的 Koopman 算子，并将其融入第三章得到的算法框架中，并通过实验的方式证明改进的有效性。最后，提出当前算法的

问题，给出未来展望。

第二章 基础知识

在本章中，首先讨论一下有关的背景理论与算法。介绍一下 Koopman 算子理论（Koopman Operator Theory），并讨论 Koopman 算子对于重塑强化学习（Reinforcement Learning）中使用的马尔可夫决策过程（Markov Decision Process）的重要作用。同时，对于 Koopman 算子理论与深度神经网络（Deep Neural Network）之间的关联。

2.1 Koopman 算子理论

系统的强非线性是数据驱动建模和控制领域的核心问题之一，包括基于现代强化学习框架所做的工作。Koopman 算子理论 [33] 为上述问题提出了一种解决方案。在该理论中，非线性系统动力学可以在提升到无限维度希尔伯特空间时变为线性系统，并可以通过一组新的基底对于该线性系统进行观测。升维工作已被证明对于线性化和简化某些具有挑战性的问题具有显著效果，这与机器学习领域中其他的类似努力相一致。

2.1.1 Koopman 算子与非线性系统演化

我们应该采用不同的形式对不同类型的系统中系统状态 x 进行描述。假设系统为确定性自治系统，我可以采用下面的方式，将一个离散动力系统描述为

$$\dot{x} = F(x) \tag{1}$$

其中，$\dot{x}$ 表示下一个时刻的系统状态；或者，采取不同的方式，将一个连续动力系统描述为

$$\frac{\mathrm{d}}{\mathrm{d}t}x(t) = f(x(t)) \tag{2}$$

而在处理实际问题是，我们通常考虑的是离散动力系统。所以我们就可以将连续系统通过流映射算子（Flow Map Operator）归纳为一个离散系统，系统演化如下

$$x(t+\tau) = F_\tau(x(t)) = x(t) + \int_t^{t+\tau} f(x(s))\mathrm{d}s \tag{3}$$

Koopman 算子提供了解决非线性系统控制问题的一个新的着眼点。在形式上，我们考虑一个实值向量测量函数 $g : M \to \mathbb{R}$，且都由无限维希尔伯特空间的元素组成，其中 M 是一个流形。通常，这个流形被认为是 $L^\infty(X)$, $X \subset \mathbb{R}^d$。一般情况下，函数 g 可被称为观测函数。Koopman 算子理论中指出，Koopman 算子 $\mathcal{K}$ 和 Koopman 生成器 $\mathcal{L}$ 都是作用于上述

观测函数 g 的无限维线性算子，在确定性系统中，有

$$\mathcal{K}g = g \circ F \tag{4a}$$

$$\mathcal{L}g = f \cdot \nabla g \tag{4b}$$

Koopman 生成器 $\mathcal{L}$ 和 Koopman 算子有如下的关系

$$\mathcal{L}g = \lim_{t\to 0} \frac{\mathcal{K}g - g}{t} = \lim_{t\to 0} \frac{g \circ F - g}{t} \tag{5}$$

Koopman 算子理论可以更广泛地应用于任何马尔可夫过程，但本文以随机性连续时间系统为例，此时，Koopman 算子的定义如下:

$$\mathcal{K}g = \mathbb{E}(g(X)|X_0 = \cdot)$$

$$\mathcal{L}g = \lim_{t\to 0} \frac{\mathcal{K}g - g}{t}$$

Koopman 算子将测量函数 g 沿着路径 x 向前演化如下:

$$\mathcal{K}_\tau g(x_t) := g(F_\tau(x_t)) = g(x_{t+\tau}) \tag{6}$$

其中 F 代表着系统的演化规律。更一般的, 在随机自治系统中，$\mathcal{K}$ 被如下定义为条件预测算子:

$$\mathcal{K}g(x_t) = \mathbb{E}\left[g(X_{t+\tau})|X_t = x_t\right] \tag{7}$$

在上述离散系统算子中，普遍使用 $\mathcal{K} := \mathcal{K}_1$，在本文中也照此用法。

2.1.2 Koopman 算子本征函数

上文提出，可观测函数 g 的观测，都存在于无限维的希尔伯特空间中（被称作观测空间），被如 Koopman 算子 $\mathcal{K}$ 等无限维的算子推动沿着给定非线性动力学系统演化。因此不难发现，我们可以应用 Koopman 算子理论研究，将对于非线性系统的研究，通过观测空间状态线性演化实现。

同时，由于难以捕捉无限维希尔伯特空间中所有可观测函数的演化，所以应该试图识别随着非线性系统动力学而线性演化的关键观测函数，Koopman 算子的本征函数就可以作为一组特殊的观测函数:

$$\mathcal{K}\Phi(x_k) = \lambda\Phi(x_k) = \Phi(x_{k+1}) \tag{8}$$

此时，本征函数就会成为观测空间的一组基底，由此，数学上，观测函数应当表示为这组基底的线性组合，如下:

$$g(x) = \sum_{i=1}^{n} a_i \Phi_i(x) \tag{9}$$

此时，观测空间可被称为状态字典空间，同时，空间的基底可被称为字典函数

当前，从数据中挖掘信息并获得 Koopman 本征函数是现代动力学系统研究的主流方法，被称为数据驱动（Data-Driven）的 Koopman 算子。由此，我们可以通过数据驱动的方式得到 Koopman 算子的本征函数，以得到非线性系统在高维空间中的全局线性表示。

此外，Koopman 算子已经被广泛运用于受控系统。在受控确定性离散时间系统中，我们有：

$$x' = F(x,u) \tag{10}$$

在受控连续时间系统中：

$$\frac{\mathrm{d}}{\mathrm{d}t}x = f(x(t),u(t)) \tag{11}$$

如果考虑到 $u_{k+1} = u_k$ 的控制输入，我们可以将等式 (7) 修改为：

$$g(x_{k+1},u_{k+1}) = \mathcal{K}g(x_k,u_k) \tag{12}$$

其中 $x \in \mathbb{R}^n, u \in \mathbb{R}^m$，且 $g : \mathbb{R}^{n+m} \mapsto \mathbb{R}^d$。

2.2 强化学习

2.2.1 马尔可夫决策过程与贝尔曼方程

马尔可夫决策过程是具有马尔可夫性质的随机过程。

随机过程是指研究对象是随时间演变的随机现象，在随机过程中，随机现象在某一时刻 t 的取值是一个向量随机变量，可以用 S_t 表示，所有可能的状态组成状态空间 $\mathcal{S}$。我们将已知所有历史状态 $(S_1,\ldots,S_t)$ 时，某一时刻 t 的状态 S_t 发生的概率用 $P(S_{t+1}|S_1,\ldots,S_t)$ 表示。而马尔可夫性质则表示，已知当前时刻状态 S_t 时，下一时刻状态 S_{t+1} 仅与 S_t 有关，用 $P(S_{t+1}|S_t)$ 表示。而从前一状态经过随机进入下一状态的过程被称为状态转移。

在下文中，我们假设存在一个无限时域的马尔可夫决策过程。我们假设代理跟随随机策略 $\pi(u|x)$，表示在已知状态 x 的情况下，采取某个特定动作 u 的可能性。由此，在离散时间系统中，状态价值函数定义为：

$$V^{\pi}(x) = \mathbb{E}\left[\sum_{k=0}^{\infty}\gamma^k r(x_k,u_k)|\pi, x_0 = x\right] \tag{13}$$

其中，$r(x_k,u_k)$ 表示智能体得到的奖励；$\gamma \in [0,1]$ 表示折扣率，用于避免在无限时域时无限大的奖励。

在这里我们强调强化学习与控制领域的结合，所以本文中考虑使用线性二次最优控制问题。线性二次最优控制是十分经典的控制领域问题，其中线性代表研究的系统动态可以用一组线性微分方程表示，而其成本为二次泛函。形式上，我们考虑有限时间长度，离散时间的 LQR，假设离散时间线性系统：

$$x_{k+1} = Ax_k + Bu_k$$

其性能指标为：

$$c(x_k,u_k)=x_k^TQx_k+u_k^TRu_k \tag{14}$$

将系统线性二次性能指标的相反数作为智能体探索时的奖励:

$$r(x_k,u_k)=-c(x_k,u_k) \tag{15}$$

并改写贝尔曼方程可以得到：

$$V^{\pi}(x)=\mathbb{E}\left[\sum_{k=0}^{\infty}-\gamma^k c(x_k,u_k)|\pi,x_0=x\right] \tag{16}$$

2.2.2 演员-评委强化学习算法

演员-评论家（Actor-Critic）强化学习算法是一种结合价值学习和策略学习的强化学习算法，下面分三步介绍 Actor-Critic 强化学习算法。

首先介绍策略学习，在策略学习中，评价一个策略的方法是评估策略在环境中获得的回报的期望，此时目标函数为：

$$J(\theta)=\mathbb{E}_S\left[V_{\pi}(S)\right] \tag{17}$$

其中，V_{π} 代表状态价值函数，可以看到在这个目标函数中，求得了状态价值函数有关状态的期望，所以只涉及到策略 π 的信息。同时易知，这个目标函数该期望越大，表明该策略越好。

注意，在本文中，对于强化学习的研究只涉及深度强化学习，所以在这里使用深度神经网络的方式对策略网络进行测试。所以，问题转化为如下的一个最大化问题：

$$\max_{\theta}J(\theta) \tag{18}$$

很明显，我们只需要使用梯度上升的方式迭代策略网络。参数更新如下：

$$\theta_{new}=\theta_{old}+\beta\cdot\nabla_{\theta}J(\theta_{now}) \tag{19}$$

其中，β 代表学习率，是需要调整的超参数，梯度的具体表达式如下：

$$\nabla_{\theta}J(\theta_{now})\triangleq\frac{\partial J(\theta)}{\partial\theta} \tag{20}$$

以上算法被称为策略梯度

在策略学习中，我们想要得到的是当前策略能得到的价值函数，表示如下：

$$V^{\pi}(s_t)=\mathbb{E}\left[\sum_{k=t}^{N}\gamma^{k-t}\cdot r_k\right] \tag{21}$$

在深度强化学习中，我们通过深度神经网络对价值进行近似，算法上，应该使得上述价值网络逼近真实价值函数。一般使用时间差分（Temporal Differencial）算法来更新价值函数，根据贝尔曼方程：

$$V^{\pi}(s_t) = \mathbb{E}\left[r_t + \gamma V^{\pi}(s_{t+1})\right] \tag{22}$$

等式左边为价值函数处于时间步 t 时对于价值的估计，等式右边是对于左边的一个无偏估计，其中包含的 r_t 项使得等式右边比左边更加精确，所以应当使得价值网络逼近等式右边的值。参数更新如下：

$$Q^{new}(s_t, a_t) \leftarrow (1-\alpha)Q^{now}(s_t, a_t) + \alpha[r_t + \gamma Q^{now}(s_{t+1}, a_{t+1})] \tag{23}$$

最后，通过交替执行策略优化和价值优化，可以使得两者的性能逐渐同步提升。策略网络可以获得更好的状态价值，价值网络可以实现更好的价值估计。最后取用策略网络作为最优控制的策略函数就可以完成 Actor-Critic 算法在最优控制问题上的应用。

2.3 神经网络

本文中将使用简单的神经网络对函数进行近似，在本节中将对于神经网络进行一些简单的介绍。

2.3.1 简单架构

神经网络架构和人类大脑中的神经细胞很像，它们相互连接组成了一个复杂的神经细胞网络状的结合体，可以发送电信号来交换信息。而人工神经网络由人工神经细胞构成，以相似的结构组合起来，共同合作以求解决实际问题。

基本的神经网络有三层 [34]：

1. 输入层：数据通过输入层经过处理进入网络；
2. 隐藏层：从输入层或其他隐藏层获取数据并计算输出；
3. 输出层：给出最后的结果，一般具有非线性；

具有上述基本层的网络叫全连接层，下面介绍全连接层 [35]：令输入向量 $x \in \mathbb{R}^d$，神经网络的一个层把 x 映射为 $x' \in \mathbb{R}^{d'}$。全连接层定义如下：

$$x' = \sigma(z), \qquad z = Wx + b \tag{24}$$

其中，权重矩阵 $W \in \mathbb{R}^{d' \times d}$ 和偏置向量 $b \in \mathbb{R}^{d'}$ 是该层的参数，需要从数据中学习；$\sigma(\cdot)$ 是激活函数，比如 softmax 函数、sigmoid 函数，ReLU（Rectified Linear Unit）函数。最常用的激活函数是 ReLU。定义如下：

$$\mathrm{ReLU}(z) = [(z_1)_+, (z_2)_+, \cdots, (z_{d'})_+,]^T \tag{25}$$

上面的 $[z_i]_+ = \max(z_i, 0)$。我们称上面的结构为全连接层（Fully Connected Layer）。如果把全连接层当作基本部件，然后像搭积木一样搭建一个全连接神经网络，也叫多层感知机（Multi-Layer Perceptron, MLP）

2.3.2 反向传播

线性模型和神经网络的训练都可以描述成一个优化问题，设全连接层的权重参数为 $\omega^{(1)}, \cdots, \omega^{(l)}$ 我们希望求解下面这样的一个优化问题 [35]：

$$\min_{\omega^{(1)}, \cdots, \omega^{(l)}} L(\omega^{(1)}, \cdots, \omega^{(l)}) \tag{26}$$

其中 L 表示损失函数。对于这样一个无约束的最小化问题，最常使用的方法是梯度下降（Gradent Descent, GD）和随机梯度下降（Stochastic Gradient Descent, SGD），本节主要介绍使用 SGD 求解最小化问题。

假设目标函数可以写成 n 连加的形式：

$$L(\omega^{(1)}, \cdots, \omega^{(l)}) = \frac{1}{n} \sum_{j=1}^{n} F_j(\omega^{(1)}, \cdots, \omega^{(l)}) \tag{27}$$

函数 F_j 隐含第 j 个训练样本 (x_j, y_j)。每次从集合 $\{1, 2, \cdots, n\}$ 中随机抽取一个数，记作 j。当前的参数只为 $\omega_{now}^{(1)}, \cdots, \omega_{now}^{(j)}$，计算此处的梯度，SGD 算法的迭代过程为

$$\omega_{new}^{(i)} \leftarrow \omega_{now}^{(i)} - \alpha \cdot \underbrace{\nabla \omega^{(i)} F_j \left(\omega_{now}^{(1)}, \cdots, \omega_{now}^{(l)} \right)}_{\text{随机梯度}}, \quad \forall i = 1, \cdots, l. \tag{28}$$

下面我们介绍反向传播（Backpropagation，BP），以全连接神经网络为例，简单介绍反向传播的原理。全连接神经网络（忽略偏移量 b）定义如下：

$$\begin{aligned} x^{(1)} &= \sigma_1(W^{(1)} x^{(0)}), \\ x^{(2)} &= \sigma_2(W^{(2)} x^{(1)}), \\ &\vdots \\ x^{(l)} &= \sigma_l(W^{(l)} x^{(l-1)}), \end{aligned}$$

神经网络的输出 $x^{(l)}$ 是神经网络做出的预测。设向量 y 为真实标签，函数 H 为交叉熵，实数 z 为损失函数：

$$z = H(y, x^{(l)}) \tag{29}$$

要做梯度下降更新参数 $W^{(1)}, \cdots, W^{(l)}$，我们需要计算损失函数 z 关于每一个变量的梯度：

$$\frac{\partial z}{\partial W^{(1)}}, \frac{\partial z}{\partial W^{(2)}}, \cdots, \frac{\partial z}{\partial W^{(l)}} \tag{30}$$

现在可以用链式法则做反向传播，计算损失函数 z 关于神经网络参数的梯度。具体地，首先求出梯度 $\frac{\partial z}{\partial x^{(l)}}$。然后做循环，从 $i=l,\cdots,1$，依次执行如下操作：根据链式法则可得损失函数 z 关于参数 $W^{(i)}$ 的梯度：

$$\frac{\partial z}{\partial W^{(i)}}=\frac{\partial x^{(i)}}{\partial W^{(i)}}\cdot\frac{\partial z}{\partial x^{(i)}} \tag{31}$$

此梯度用于更新参数 $W^{(i)}$。根据链式法则可得损失函数 z 关于参数 $x^{(i-1)}$ 的梯度:

$$\frac{\partial z}{\partial x^{(i-1)}}=\frac{\partial x^{(i)}}{\partial x^{(i-1)}}\cdot\frac{\partial z}{\partial x^{(i)}} \tag{32}$$

这个梯度被传播到下面一层（即第 $i-1$ 层）, 继续循环。

综上所述，只要知道损失函数 z 关于 $x^{(i)}$ 的梯度，就能求出 z 关于 $W^{(i)}$ 和 $x^{(i-1)}$ 的梯度。

第三章 Koopman 算子辅助的最大熵强化学习

贝尔曼方程及其连续形式汉密尔顿-雅可比-贝尔方程在强化学习和控制理论中普遍存在。然而，这些方程对于具有高维状态和非线性的系统来说，很快就变得难以处理。本章将讨论数据驱动的 Koopman 算子与马尔可夫决策过程之间的联系，并介绍借由此开发的两种新的强化学习算法来解决上述的问题。

技术上，我们可以利用 Koopman 算子技术将非线性系统提升到新的坐标中，其中动力学变得近似线性。特别是，Koopman 算子能够通过提升坐标中的线性动力学来捕获给定系统的价值函数的时间演化的期望。

3.1 控制有关的数据驱动 Koopman 算子

数据驱动的 Koopman 算子是一种从系统数据中学习系统动力学行为的方法。它的主要思想是利用系统的观测数据来构建 Koopman 算子的近似，而无需事先了解系统的动力学方程。具体来说，给定系统的观测数据序列 $x_0, x_1, \ldots, x_N$，其中 x_i 是系统在时间 t_i 的状态观测值，数据驱动的 Koopman 算子可以通过以下步骤获得：

3.1.1 数据驱动的 Koopman 算子

下面介绍两种获取数据驱动的 Koopman 算子的方式。

动态模态分解

动态模态分解 [36] (Dynamic Mode Decomposition, DMD) 是一种基于数据的模型约简技术，用于从系统观测数据中提取动力学模式。给定系统的观测数据序列 $x_0, x_1, \ldots, x_N$，DMD 可以分解系统的状态演化矩阵 A，并提取系统的动态模态。具体步骤如下：构建数据矩阵 X 和下一时刻数据矩阵 X'。对数据矩阵 X 和 X' 进行奇异值分解 (SVD)，得到矩阵 U, Σ, V。估计系统状态演化矩阵 A。计算系统的动态模态，即 A 的特征向量。DMD 可以用于系统预测、模型约简和特征提取等应用，在流体力学、结构动力学和气象学等领域得到广泛应用。

扩展动态模态分解

扩展动态模态分解 [37] (Extended Dynamic Mode Decomposition, EDMD) 是 DMD 的一种扩展形式，用于处理非线性系统的观测数据。与传统的 DMD 相比，EDMD 引入了非线性特征映射，以便处理非线性系统的演化。具体步骤如下：构建非线性特征映射，将观测数据映射到高维特征空间。在高维特征空间中构建数据矩阵 X 和下一时刻数据矩阵 X'。对

数据矩阵 X 和 X' 进行奇异值分解 (SVD)，得到矩阵 U, Σ, V。估计系统状态演化矩阵 A。计算系统的动态模态，即 A 的特征向量。EDMD 通过引入非线性特征映射，可以更好地处理非线性系统的观测数据，并提取系统的动态模态。

3.1.2 动作加权的 Koopman 算子

我们现在关注如何通过 Koopman 算子在时间上推进观测基底函数，已知当前状态和动作，也就是说，找到下面的一个映射：

$$(x, u) \mapsto \phi(\dot{x}) \tag{33}$$

我们要求观测函数近似地覆盖每个控制变量 u 的价值函数的有限维的 Koopman 不变子空间，因此存在矩阵 $K^u \in \mathbb{R}^{d_x \times d_x}$，使得 $K^u \phi(x) = \phi(\dot{x})$。

给定轨迹数据和状态字典空间 ϕ，为每个动作 $u \in \mathcal{U}$ 构建如上的 Koopman 矩阵 K^u。为此，采取类似于 SINDYc [38] 中描述的方法，其中状态和动作的字典空间用于预测下一个状态，即 $\Theta(x, u) \mapsto \dot{x}$ 而这里的方法不同的地方有两点。首先，不是试图预测下一个状态 x，而是试图预测下一个字典函数值 $\phi(x)$。其次，为了让状态字典空间覆盖每个 u 的 Koopman 不变子空间，状态-动作字典在状态和动作上是可分离的。

这里，将状态-动作字典空间建模为乘法可分离的，就像 $\psi(x) \otimes \phi(x)$ 然后进一步假设存在线性映射 $\psi(x) \otimes \phi(x) \mapsto \phi(\dot{x})$。下面描述如何为任何的 u 构建矩阵 K^u。

令 $\phi : \chi \mapsto \mathbb{R}^d_x$ 为状态的特征映射（ϕ 的每一个分量都是一个可观测函数），并且令 $\psi : \mathcal{U} \mapsto \mathbb{R}^d_u$ 为控制动作的特征映射。算法要求对于所有 $u \in \mathcal{U}$ 得到 Koopman 算子 $\mathcal{K}^u$ 的有限维逼近。令 $\mathcal{T}_K \in \mathbb{R}^{d_x \times d_x \times d_u}$ 为如图 [ref] 所示的三维向量。对于任何 u，如下定义：

$$K^u[i, j] = \sum_{z=1}^{d} \mathcal{T}_K(i, j, k) \phi(u)[z] \tag{34}$$

形式上，K^u 是沿着张量 $\mathcal{T}$ 第三维的向量积，同时，K^u 作为对于 Koopman 算子 $\mathcal{K}^u$ 的有限维近似。因此，我们可以通过学习 $\mathcal{T}_K$，来最小化字典函数 ϕ 的误差：

$$\min_{\mathcal{T}_K} \sum_{i=1}^{N} ||K^{u_i} \phi(x_i) - \phi(x'_i)||^2 \tag{35}$$

我们可以重写上述目标，使其成为一个规则的多变量线性回归问题，将 $\mathcal{T}_K$ 重新排列为一个大小为 $\mathbb{R}^{d_x \times d_x d_u}$ 的二维矩阵。令 $M \in \mathbb{R}^{d_x \times d_x d_u}$，其中 $M[i, :] \in \mathbb{R}^{d_x \cdot d_u}$ 是堆叠二维矩阵 $\mathcal{T}_K[i, :, :]$ 所得到的向量，如图 [3.1] 所示。令 $\psi(u) \otimes \phi(x) \in \mathbb{R}^{d_x d_u}$ 为克罗内克积（Kronecker Product），所以有：

$$K^u \phi(x) = M(\psi(u) \otimes \phi(x)) \tag{36}$$

从而，优化问题变成了常规的线性规划问题：

$$\min_{M} \sum_{i=1}^{N} ||M(\phi(u) \otimes \phi(x_i)) - \phi(\dot{x}_i)||^2 \tag{37}$$

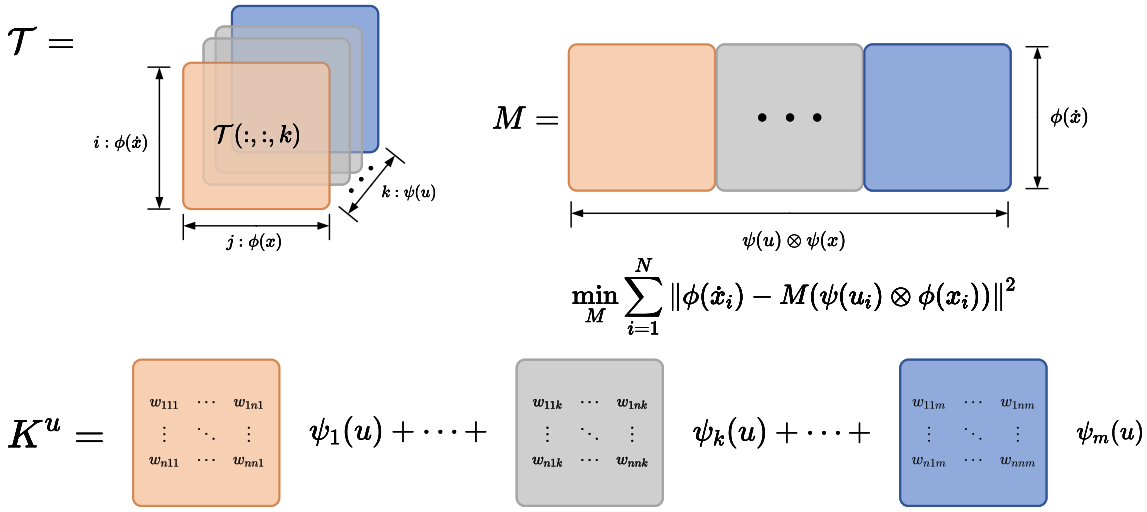

图 3.1 Koopman 张量估计

算法见算法 1 。

只要我们计算出了 M，我们就可以通过将 M 转换回为三维张量 $\mathcal{T}_K$ 的方式为任意系统执行的 $u \in \mathcal{U}$ 得到标准的 Koopman 算子，同时也得到了上述对 $\mathcal{K}^u$ 的有限维近似 $K^u \in \mathbb{R}^{d_x \times d_x}$

算法 1 Koopman 张量估计

输入： 状态特征映射 $\phi : \chi \mapsto \mathbb{R}^{d_x}$，控制特征映射 $\psi : \mathcal{U} \mapsto \mathbb{R}^{d_u}$ 以及一个样本集合：$\{(x_i, u_i)\}_{i=0}^{N}$

1: 从等式 (37) 中解出 $\hat{M}$

2: 以 Fortran 风格将 M 转换成 $\mathcal{T}_K$

3.2 最大熵强化学习最优控制问题

考虑强化学习在最优控制领域的应用，在实践中最大的问题有以下两点：首先，非常高的样本复杂度：使得智能体在探索中遇到困难；其次，缓慢、不稳定的收敛过程：探索和利用的权衡是策略优化的重点，特别是在最优控制中，不合适的样本利用和探索会使得策略过早的收敛到不良的局部最优值。本文中使用最大熵强化学习算法，通过在训练目标中合理加入熵正则化项。经过训练，策略可以最大化预期回报和熵。

3.2.1 熵正则策略优化

粗略的说，熵是一个表示随机变量随机程度的量。具体地说，如果一个随机变量的概率分布很集中，那么这个随机变量的熵就低；如果一个随机变量的概率分布比较分散，那么这个随机变量的熵就高。

令 x 是具有概率质量密度函数的随机变量 P。H 的熵是根据 x 的分布得出的：

$$H(P) = \mathop{\mathbb{E}}_{x\sim P}[-\log P(x)]$$

在熵正则强化学习中，代理在每个时间步获得与该时间步的策略熵成比例的奖金。由此，目标策略 [39] 变为：

$$\pi^* = \arg\max_{\pi} \mathop{\mathbb{E}}_{\tau\sim\pi}\left[\sum_{t=0}^{\infty}\gamma^t\left(r(s_t,a_t,s_{t+1}) + \alpha H(\pi(\cdot|s_t))\right)\right] \tag{38}$$

其中 $\alpha > 0$ 是权衡参数。我们现在可以在如上假定中给出略有不同的价值函数。V^π 更改为包含每个时间步的熵奖励：

$$V^\pi = \mathop{\mathbb{E}}_{\tau\sim\pi}\left[\sum_{t=0}^{\infty}\gamma^t\left(r(s_t,a_t,s_{t+1}) + \alpha H(\pi(\cdot|s_t)) \mid s_0 = s\right)\right] \tag{39}$$

Q^π 更改为包括除了第一个时间步以外的每个时间步的熵奖励：

$$Q^\pi(s,a) = \mathop{\mathbb{E}}_{\tau\sim\pi}\left[\sum_{t=0}^{\infty}\gamma^t r(s_t,a_t,s_{t+1}) + \alpha\sum_{t=1}^{\infty}\gamma^t H(\pi(\cdot|s_t)) \mid s_0 = s, a_0 = a\right] \tag{40}$$

通过这些定义，V^π 和 Q^π 通过以下方程联系起来：

$$\begin{aligned} V^\pi(s) &= \mathop{\mathbb{E}}_{\dot{s}\sim P,\dot{a}\sim\pi}[r(s,a,\dot{s}) + \gamma(Q^\pi(\dot{s},\dot{a}) + \alpha H(\pi(\cdot|\dot{s})))] \\ &= \mathop{\mathbb{E}}_{\dot{s}\sim P}[r(s,a,\dot{s}) + \gamma V^\pi(\dot{s})] \end{aligned} \tag{41}$$

标准的强化学习最大化奖励之和的期望，但是在加入熵正则方法的强化学习中，将会考虑一个更加一般的最大熵目标：

$$J(\pi) = \sum_{t=0}^{T}\mathbb{E}[r(s_t,a_t) + \alpha\mathcal{H}(\pi(\cdot|s_t))] \tag{42}$$

权重参数 α 决定了熵项对于奖励的相对重要性，从而控制了最优策略迭代中的的随机性。

3.2.2 Soft Actor-Critic 强化学习最优控制算法

下面，介绍如何通过策略迭代公式设计 SAC 算法 [7]，我们将从推导熵策略迭代（Soft Policy Iteration）开始，这是一种在应用了熵正则的方法下学习最优策略的通用算法，在最大熵框架中交替进行策略评估和策略改进。

在策略评估步骤中，我们希望根据等式 (42) 中的最大熵目标计算策略 π 的值。对于固定策略，可以从任何函数 $Q : \mathcal{S}\times\mathcal{A}\mapsto\mathbb{R}$ 开始，并且使用下面的修改过的贝尔曼方程迭代计算熵动作价值：

$$Q(s_t,a_t) = r(s_t,a_t) + \gamma\mathbb{E}_{s_{t+1}\sim p}[V(s_{t+1})] \tag{43}$$

在策略改进步骤中，策略将逼近于新的熵动作价值函数的指数。如下的这种特定的设定可以保证策略的改进 [7,40]：

$$\pi_{new} = \arg\min_{\pi' \in \Pi} \mathrm{D}_{KL}\left(\pi'(\cdot|s_t) \middle\| \frac{\exp(Q^{\pi_{old}}(s_t,\cdot)}{Z^{\pi_{old}}(s_t)}\right) \tag{44}$$

尽管该算法将可证明地找到最优解，但我们只能在表格情况（有限个状态和动作）下以其精确形式执行它。因此，我们接下来将对连续域进行估计，其中我们需要依靠函数逼近器来表示动作价值函数，这种近似的做法产生了一种新的实用算法，称为 Soft Actor-Critic 算法。

如上所述，大型连续域要求我们推导出熵策略迭代的实际逼近。为此，我们将对动作价值函数和策略使用函数逼近器，并且交替使用随机梯度下降（Stochastic Gradient Descent）来优化这两个网络。我们将考虑参数化状态价值函数 $V_\psi(s_t)$、熵动作价值函数 $Q_\theta(s_t,a_t)$ 和策略 $\pi_\phi(a_t|s_t)$。这些网络的参数是 ψ,θ 和 ϕ。下面介绍这些参数向量的更新规则。

设定如下熵状态函数训练目标，以最小化平方残差：

$$J_V(\psi) = \mathop{\mathbb{E}}_{s_t\sim D}\left[\frac{1}{2}(V_\psi(s_t) - \mathbb{E}_{a_t\sim\pi_\phi}\left[Q_\theta(s_t,a_t) - \log\pi_\phi(a_t,|s_t)\right])^2\right] \tag{45}$$

其中，D 是先前采样状态和动作的概率分布，或者是经验回放。等式 (45) 的梯度可以用下面的无偏估计表示：

$$\hat{\nabla} J_V(\psi) = \nabla_\phi V_\phi(s_t)(V_\phi(s_t) - Q_\theta(s_t,a_t) + \log\pi_\phi(a_t|s_t)) \tag{46}$$

其中，动作是根据当前策略采样的，而不是从经验回放中抽取。可以如下训练熵动作价值函数参数来最小化熵贝尔曼残差：

$$J_Q(\theta) = \mathbb{E}_{(s_t,a_t)\sim D}\left[\frac{1}{2}(Q_\theta(s_t,a_t) - \hat{Q}(s_t,a_t))^2\right] \tag{47}$$

其中

$$\hat{Q}(s_t,a_t) = r(s_t,a_t) + \gamma\mathbb{E}_{s_{t+1}\sim p}\left[V_{\overline{\psi}}(s_{t+1})\right] \tag{48}$$

同样可以用随机梯度下降优化：

$$\hat{\nabla}_\theta J_Q(\theta) = \nabla_\theta Q_\theta(a_t,s_t)(Q_\theta(s_t,a_t) - r(s_t,a_t) - \gamma V_{\overline{\phi}}(s_{t+1})) \tag{49}$$

更新利用了目标值网络 $V_{\overline{\psi}}$，其中 $\overline{\psi}$ 在更新参数时加权更新，这种方式已可以被证明稳定训练。最后，可以通过直接最小化等式 (44) 中的 KL 散度（Kullback-Leibler Divergence）来学习策略参数：

$$J_\pi(\phi) = \mathbb{E}_{s_t\sim D}\left[D_{KL}\left(\pi_\phi(\cdot|s_t) \middle\| \frac{\exp(Q_\theta(s_t,\cdot)}{Z_\theta(s_t)}\right)\right] \tag{50}$$

首先使用神经网络变换来参数化策略：

$$\mathbf{a}_t = f_\phi(\epsilon_t;\mathbf{s}_t) \tag{51}$$

其中 t 是输入噪声向量，从一些固定分布中采样，例如球面高斯。我们现在可以将等式 (50) 中的目标重写为：

$$J_\pi(\phi) = \mathbb{E}_{s_t\sim D,\epsilon_t\sim N}\left[\log\pi_\phi(f_\phi(\epsilon_t;s_t)|s_t) - Q_\theta(s_t, f_\phi(\epsilon_t;s_t))\right] \tag{52}$$

我们可以近似等式 (52) 的梯度：

$$\hat{\nabla}_\phi J_\pi(\phi) = \nabla_\phi \log\pi_\phi(\mathbf{a}_t|\mathbf{s}_t) + (\nabla_{\mathbf{a}_t}\log\pi_\phi(\mathbf{a}_t|\mathbf{s}_t) - \nabla_{\mathbf{a_t}}Q(\mathbf{s}_t,\mathbf{a}_t))\nabla_\phi f_\phi(\epsilon_t;\mathbf{s}_t) \tag{53}$$

其中，a_t 由 $f_\phi(\epsilon_t;\mathbf{s}_t)$ 评估。

完整的算法见算法 **2** 。

算法 2 Soft Actor-Critic 强化学习算法

初始化参数向量 $\psi,\overline{\psi},\theta,\phi$
for 每一次迭代 **do**
  for 每一个时间步 **do**
    $\mathbf{a}_t \sim \pi_\phi(\mathbf{a}_t|\mathbf{s}_t)$
    $\mathbf{s}_{t+1} \sim p(\mathbf{s}_{t+1}|\mathbf{s}_t,\mathbf{a}_t)$
    $D \leftarrow D \cup (\mathbf{s}_t,\mathbf{a}_t,r(\mathbf{s}_t,\mathbf{a}_t),\mathbf{s}_{t+1})$
  end for
  for 每一次梯度 **do**
    $\psi \leftarrow \psi - \lambda_V\hat{\nabla}_\phi J_V(\psi)$
    $\theta_i \leftarrow \theta_i - \lambda_Q\hat{\nabla}_{\theta_i}J_Q(\phi)$ for $i\in 1,2$
    $\phi \leftarrow \phi - \lambda_\pi\hat{\nabla}_\phi J_\pi(\phi)$
    $\overline{\psi} \leftarrow \tau\psi + (1-\tau)\overline{\psi}$
  end for
end for

下面，介绍如何通过修改 Soft Actor-Critic 框架 [7]，以合并来自 Koopman 算子的信息来限制探索空间的。

这里使用与 Soft Actor-Critic 论文 [53] 相同的损失函数和类似的符号，我们首先指定熵价值函数损失：

$$J_V(\psi) = \mathop{\mathbb{E}}_{s_t\sim D}\left[\frac{1}{2}(V_\psi(s_t) - \mathbb{E}_{a_t\sim\pi_\phi}\left[Q_\theta(s_t,a_t) - \log\pi_\phi(a_t,|s_t)\right])^2\right] \tag{54}$$

在这里，介绍有关 Koopman 熵价值函数的新定义：

$$V_\psi(x) = \psi^T\phi(x) \tag{55}$$

其中，ω 是字典函数的系数向量，表示价值函数在是字典函数也就是观测函数的线性组合。也就是说，价值函数的评估是集中在 Koopman 升维后的观测空间中进行的。同时，我们知

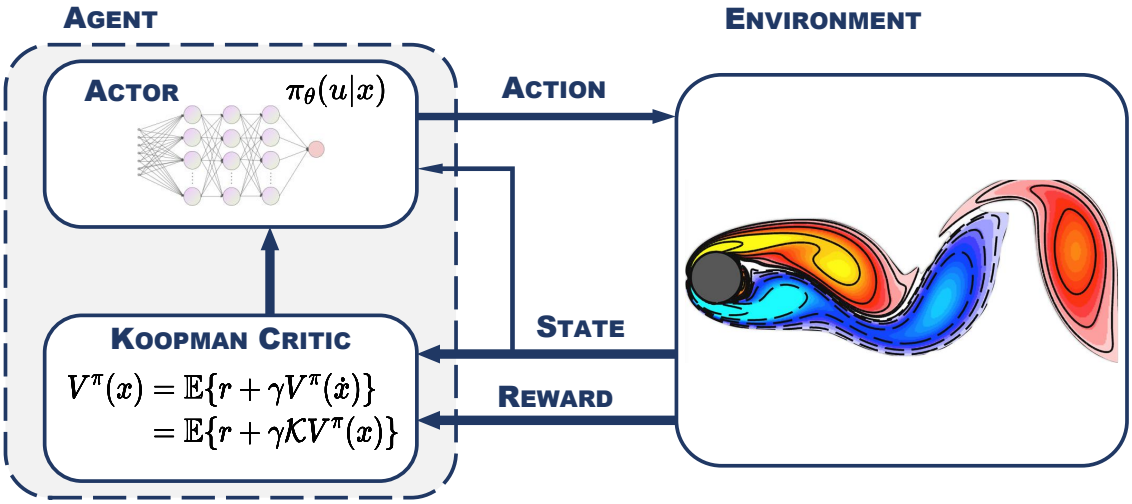

图 3.2 Soft Actor Koopman-Critic

道，合理的观测空间应该是由一组完备的观测函数所构建的，所以在价值评估的准确性上是有理论支撑的。

下面，介绍如何改动动作价值函数的目标函数：

$$J_Q(\theta) = \mathbb{E}_{(x,u)\sim D}\left[\frac{1}{2}\left(Q_\theta(x,u) - \hat{Q}(x,u)\right)^2\right] \tag{56}$$

其中，训练目标融合了 Koopman 算子，被定义为：

$$\hat{Q}(s_t,a_t) = r(s_t,a_t) + \gamma \mathbb{E}_{s_{t+1}\sim p(\cdot|s_t,a_t)}\left[V_{\overline{\omega}}(s_{t+1})\right] \tag{57}$$

在这里，考虑价值预测向量 $\overline{\omega}$ 和 Koopman 算子，将上式记为：

$$\hat{Q}(s_t,a_t) = r(s_t,a_t) + \gamma \overline{\omega}^T K^u \phi(s_t) \tag{58}$$

最后，策略目标函数并没有更改：

$$J_\pi(\nu) = \mathbb{E}_{s\sim D}\left[D_{KL}\left(\pi_\nu(\cdot|s)\middle\|\frac{\exp(Q_\theta(s,\cdot))}{Z_\theta(s)}\right)\right] \tag{59}$$

通过上述修改，实际算法在表述上与算法 2一致。而不同之处已经在上述公式中解释清楚了。

Koopman 辅助的最大熵强化学习以 Soft Actor Koopman-Critic 为例，创建了 Koopman Critic，其本质是流行的 Soft Actor-Critic 算法的 Koopman 变体。

Koopman Critic 接收状态和奖励作为原始的非线性动态，然后将这些动态提升到向量空间，在那个空间里，系统状态可以通过 Koopman 算子线性地推进。然后将此批评反馈给 Actor，Actor 据此计算要在环境中执行的动作。整体的框架如图 3.2所示。

3.3 本章小结

本章主要介绍一种用于非线性系统最优控制系统的，基于 Koopman 算子辅助强化学习训练的算法。该算法的研究对象主要是第二章中提及的受控非线性系统，训练算法为最大熵强化学习。

同时，通过 Koopman 算子理论，将最大熵强化学习中的价值函数，在贝尔曼方程中进行代换的方式，得到贝尔曼方程中价值函数有关时间序列的期望，这样做的好处是提高算法对于非线性系统的适应性，同时保留了 Koopman 算子对于非线性系统动态的洞察，保留了输入输出的可解释性。

第四章 深度 Koopman 算子非线性系统最大熵强化学习

在本章中，首先介绍一个端到端的神经网络学习框架，用于替代上文介绍的其他数据驱动的 Koopman 算子，由于该神经网络架构额外的对于高维度系统动力学信息更好地把握，可以期望其得到更精准的预测。同时，将其与 Koopman 辅助的强化学习结合，得益于更加精确的价值函数估计，可以使得最大熵强化学习效果更好，训练更快。

4.1 深度 Koopman 算子

4.1.1 深度嵌入函数

我们首先将嵌入函数分为两部分：在不失一般性的情况下，我们假设第一项仅与系统状态相关，$g(x,u)=[g_x(x,u);g_u(x,u))]$ 所以我们得到 $g_x(x,u)=g_x(x)$。基于 Koopman 算子的定义，将 Koopman 算子 $\mathcal{K}$ 拆分成两个矩阵，则可以得出：

$$g_x\left(x_{k+1}\right)=K_{xx}g_x\left(x_k\right)+K_{xu}g_u\left(x_k,u_k\right) \tag{60}$$

对于嵌入函数的第一部分，我们使用神经网络 θ 来参数化 $g_x(x,k)$，将原始状态和网络编码连接起来在一起:

$$z_k=g_x\left(x_k\right)=\begin{bmatrix} x_k \\ g_\theta\left(x_k\right) \end{bmatrix} \tag{61}$$

其中 z_k 是升维系统（这里也叫嵌入空间）状态，$g_\theta:\mathbb{R}^n\mapsto$ 是参数神经网络。这种设计的优点是可以通过以下方式轻松恢复原始状态：

$$x_k=Cz_k \tag{62}$$

其中，矩阵 $C\in\mathbb{R}^{n\times(n+d)}$ 的一个样板如下：

$$C=[I_n \quad 0] \tag{63}$$

这样，我们就可以保留成本函数的形式进行进一步的控制。

对于嵌入函数的第二部分，在深度 Koopman 算子理论中，有三个版本来表示 $g_u(x_k,u_k)$。

第一个版本被称为深度 KoopmanU（Deep KoopmanU with Control, DKUC）算法。其将函数简化为 $g_u(x_k,u_k)=u_k$，然后演化方程退化为 $g_x(x_{k+1})=K_{xx}g_x(x_k)+K_{xu}u_k$。

第二个版本被称为深度 Koopman 仿射（Deep Koopman Affine with Control, DKAC）算法，它将进化函数视为控制仿射形式，使得 $g_u(x_k,u_k)=g_u(x_k)u$，并且将等式 (7) 修改为：

$$z_{k+1}=K_{xx}z_k+K_{xu}g_u(x_k)u \tag{64}$$

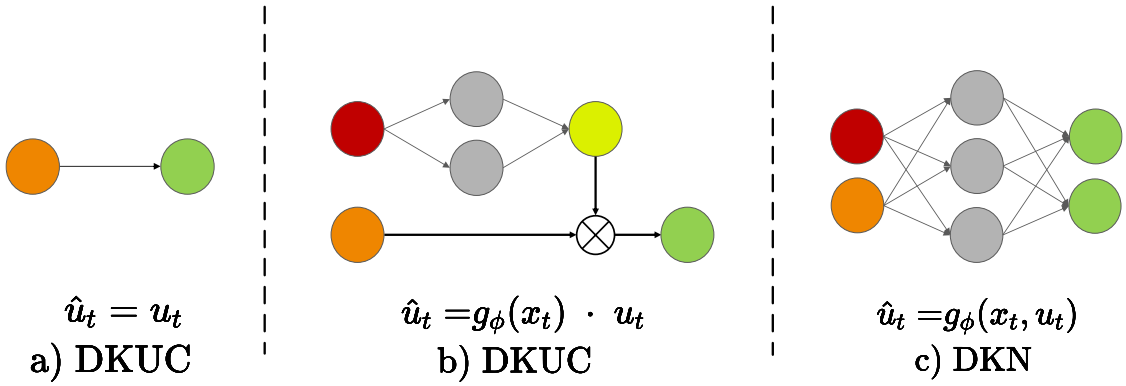

图 4.1 不同形式的控制网络示意图

对于最终版本，使用控制网络 ϕ 将函数 $g_u(x_k,u_k)$ 直接近似为 $g_u(x_k,u_k)=g_\phi(x_k,u_k)$，并且演化函数保持与等式 (7) 相同的形式。它被称为深度 Koopman 非线性（Deep Koopman Nonlinear, DKN）算法。

给定当前状态 x_t，我们可以很容易地通过前馈网络预测未来的 K 步状态。除了嵌入函数外，我们还通过一层线性网络将 Koopman 算子矩阵 K_{xx} 和 K_{xu} 参数化，即令 $K_{xx},K_{xu}=B$。然后 K 步预测如下：

$$\begin{aligned} z_{t+k+1} &= Az_{t+k} + Bg_\phi\left(x_{t+k},u_{t+k}\right), k=0,\ldots,K-1 \\ z_t &= g_x\left(x_t\right) = \begin{bmatrix} x_t \\ g_\theta\left(x_t\right) \end{bmatrix} \\ x_{t+k} &= Cz_{t+k} \end{aligned} \tag{65}$$

其中，对于 DKUC 算法，$g_\phi(x,u)=u$，对于 DKAC 算法，$g_\phi(x,u)=g_\phi(x)u$，对于 DKN 算法，$g_\phi(x,u)=g_\phi(x,u)$。网络示意如图 (4.1)

4.1.2 K 步损失函数

Koopman 的标准算法应首先选择嵌入函数，然后使用线性回归学习 Koopman 算子矩阵，这种方法仅用于单步预测。在这项工作中，我们端到端地学习嵌入函数和 Koopman 算子，并设计了用于长期预测的 K 步预测损失函数。给定数据集 $[X_i\in\mathbb{R}^{N\times n},U_i\in\mathbb{R}^{N\times m},i=0,1,2,\cdots,K]$ 我们可以计算实际嵌入状态 $Z_i=g_x(X)$，并通过方程 (65) 从初始状态 X_0 预测嵌入空间中的 K 阶状态 $[\hat{Z}_i,i=1,2,\cdots,K]$。损失函数计算如下：

$$L(\theta,\phi)=\sum_{i=1}^{K}\gamma^{i-1}\,\mathrm{MSE}\left(Z_i,\hat{Z}_i\right) \tag{66}$$

其中 是权重衰减超参数，MSE 是均方损失函数 (Mean Square Error)。K 步损失不是只考虑下面的一步预测误差，而是关注 K 步预测误差的加权和，有利于在长时间范围内进行预测。

4.2 非线性系统最大熵强化学习

上一节介绍了深度 Koopman 算子是如何通过网络架构和而在这一节中，我们介绍如何将深度 Koopman 算子和最大熵强化学习结合，以求使得我们的最优控制算法得到更好的收敛速度和训练效果。

在第三章中，介绍了动作加权的 Koopman 算子，其目的是为了更好的获取系统动作的信息，不同的动作量信息，在做单步预测时，应当在 Koopman 算子中体现出来，以求达到更好的预测能力。

在实际的控制中，动作变量可能与系统的多个变量有着联系，甚至其中的数学关系不能够用线性关系来描述，很明显，简单的加权计算并不能够表述这样的关系。这就会导致动作的轨迹在升维空间或称之为嵌入空间中会发生偏移和失真。也就简介导致了 Koopman 价值函数在这种情况下会发生误差，导致学习过程受影响。

4.2.1 深度 Koopman Critic

下面介绍深度 Koopman Critic 网络来解决上面的问题：

我们假设有嵌入网络 $g_\theta : d_x \mapsto d_z$，控制辅助网络 g_ϕ，Koopman 算子矩阵 A, B 和恢复矩阵 C。下面介绍网络架构和前馈过程：

有系统状态和输入：(x_t, u_t)，嵌入网络将系统状态提升到嵌入空间：

$$z_t = [\ x_t^T \quad g_\theta(x_t)^T\ ]^T \tag{67}$$

同样的，辅助控制网络将系统控制提升到嵌入空间：

$$\hat{u_t} = g_\phi(x_t, u_t) \tag{68}$$

这里，使用了深度 Koopman 非线性形式的辅助网络，使用了当前时刻系统状态 x_t 和系统控制 u_t 作为输入信息，在使用了神经网络参数化的策略函数的情况下，保证了可以准确捕捉动作信息与状态信息之间的关系。

下面介绍两个 Koopman 算子分量矩阵 A, B:

$$z_{t+1} = Az_t + B\hat{u}_t \tag{69}$$

可以看出，在这里我们使用了两个 Koopman 矩阵，并且取两个矩阵的和作为新的升维状态，同时考虑了状态信息和动作信息。具体网络架构如图 (A.1) 所示。

在此基础上，我们考虑如等式 (57) 中的 Koopman-Critic 函数，将 Koopman 算子和嵌入函数结合的函数 $K^u\phi(\cdot)$ 通过上述的神经网络结合，将等式 (57) 改写为：

$$\hat{Q}(s_t, a_t) = r(s_t, a_t) + \gamma\overline{\omega}^T(Az_t + B\hat{u}_t) \tag{70}$$

在上面的修改下，我们依然可以遵从算法 2 对参数网络进行迭代。

4.2.2 实验验证

在本节中，为测试算法性能，在经典的 Lorenz 环境中进行测试，这是一个标准的正则混沌基准动力学系统，由于其连续的特征值谱 [41]，通常用于评估基于 Koopman 的方法。连续时间动力学由以下方程组给出：

$$f(x,u)=\begin{bmatrix}\sigma(x_1-x_0)+u\\(\rho-x_2)x_0-x_1\\x_0x_1-\beta x_2\end{bmatrix} \tag{71}$$

其中，$\sigma=10$, $\rho=28$, $\beta=8/3$。

文中用了包括 SAC、PPO 和 LQR 在内的三种算法和本章中提出的 DSAKC 算法作对比。代码基于 CleanRL [42] 进行改进，实现上述算法在 Lorenz 环境下的学习。首先，我让四种算法在 Lorenz 环境中探索并优化了 50000 步，并记录下每 2000 步的回报，如图4.2所示。

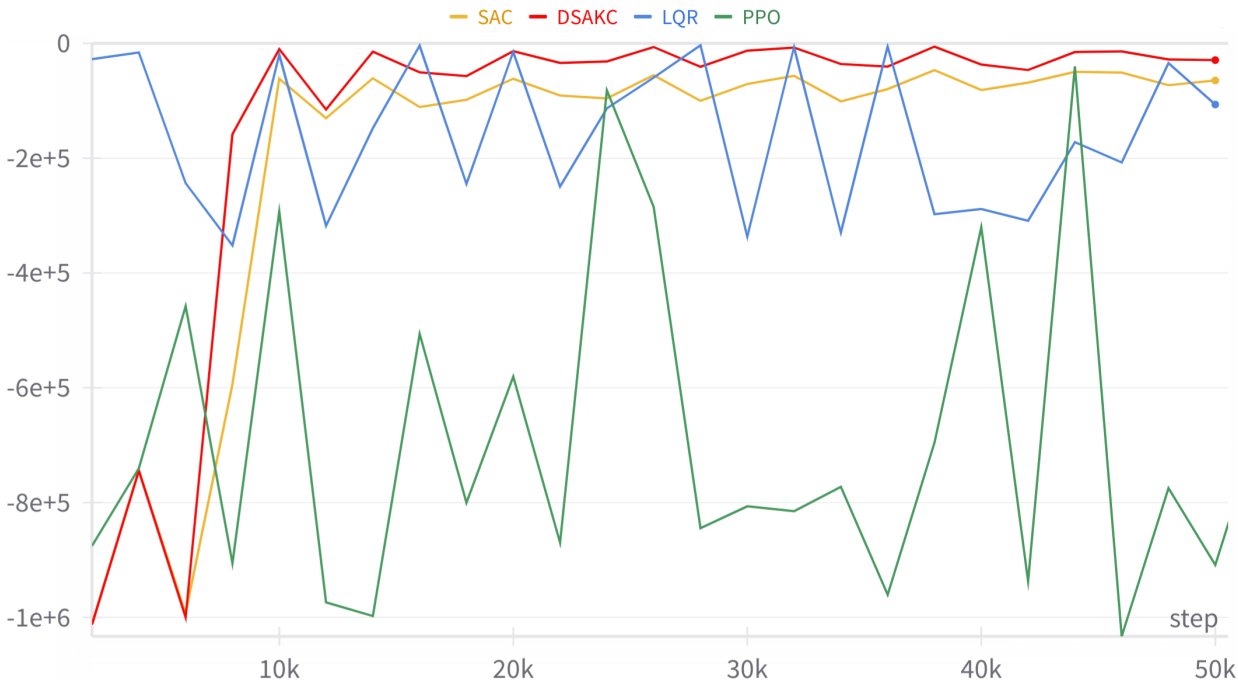

图 4.2 四种算法在 Lorenz 系统中每 2000 步回报曲线图

比较 DSAKC 算法和 SAC 算法，算法在同一个随机种子的情况下进行训练，前 4000 步随机探索，此后的回报均为策略探索得到，我们可以看到对于 SAC 而言 DSAKC 在 Lorenz 系统的控制上，有着明显的优势。

算法	PPO	LQR	SAC	DSAKC
2000 步回报	-724204	-156293	-74895	**-33071**

表 4.1 四种算法在 Lorenz 系统中运行 25 回合的平均回报

而对于其他算法，在探索的过程中，并没有表现出持续很好的效果，但是在探索的过程中，有一些策略在其回合中得到了较小的回报，下面我将用四种算法训练出的策略，在同样的 Lorenz 系统中，使用训练中表现最好的（即回报最小的策略）运行 50000 步，即，25 个回合，同时将得到的平均回报绘制成图表，如表4.1所示。

第五章 总结与展望

5.1 论文总结

在本文中，我们采用了深度强化学习中的最大熵强化学习的框架对非线性系统进行控制，其中，熵正则策略优化可以在训练的过程中，加强策略的探索，平衡数据探索和利用，在解决过早收敛的问题的同时最大化回报。

进一步的，为了解决非线性系统复杂的动力学给价值评估带来的问题，加入 Koopman 算子理论，使用 Koopman 嵌入函数，将系统动力学嵌入到一个更高维的线性空间中，并使用 Koopman 算子推进系统动力学。

在现代 Koopman 算子理论中，Koopman 算子可以使用数据驱动的方式得到，但是嵌入函数的选取仍然是难以解决的问题，在非线性系统中，根据状态信息构建的嵌入函数，往往会导致动作信息在嵌入空间中产生畸变。而深度 Koopman 算子，以及其采用的端到端的训练框架，为嵌入函数加入非线性形式的控制信息，可以有效地解决此问题。

最后，我提出深度 Koopman 算子辅助的最大熵强化学习算法，使用 Koopman 算子及其嵌入函数，对贝尔曼方程进行改写，将价值估计提升到嵌入空间进行，使得价值函数可以在高维空间上线性地推进，以提升价值估计的准确性，进而提升训练的效果，使得智能体在强非线性系统中有比较好的表现。实验证明，在具有连续谱的非线性 Lorenz 系统中，算法展现了比较好的表现，在 50000 步的训练和探索中均优于普通的深度强化学习和线性控制算法的表现。

5.2 未来展望

目前，算法主要的问题有两点：

1. 深度 Koopman 算子和最大熵强化学习，采用先后离线训练的方式进行，这样会导致样本数据的浪费问题，且无法实现真正的端到端的训练。
2. 算法实际意义不强，缺失机械臂控制实验。

未来的发展方向主要在于：

1. 深度 Koopman 算子共用经验回放，解决数据浪费的问题；实现在线训练可能。
2. 寻求与机械臂结合，发挥应对非线性系统优势，实现控制任务。

参考文献

[1] Lusch B, Kutz J N, Brunton S L. Deep learning for universal linear embeddings of nonlinear dynamics[J]. Nature Communications, 2018, 9(1):4950. Number: 1 Publisher: Nature Publishing Group.

[2] Kaiser E, Kutz J N, Brunton S L. Data-driven discovery of Koopman eigenfunctions for control[Z], February, 2021. http://arxiv.org/abs/1707.01146. arXiv:1707.01146 [math].

[3] Calderón H M, Schulz E, Oehlschlägel T, et al. Koopman Operator-based Model Predictive Control with Recursive Online Update[C]. In 2021 European Control Conference (ECC), 1543–1549, June, 2021.

[4] Retchin M, Amos B, Brunton S, et al. Koopman Constrained Policy Optimization: A Koopman operator theoretic method for differentiable optimal control in robotics[C]. September, 2023.

[5] Rozwood P, Mehrez E, Paehler L, et al. Koopman-Assisted Reinforcement Learning[C]. October, 2023.

[6] Lillicrap T P, Hunt J J, Pritzel A, et al. Continuous control with deep reinforcement learning[Z], July, 2019. http://arxiv.org/abs/1509.02971. arXiv:1509.02971 [cs, stat].

[7] Haarnoja T, Zhou A, Abbeel P, et al. Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor[J]. CoRR, 2018, abs/1801.01290. ArXiv: 1801.01290.

[8] Brunton S L, Kutz J N. Data-driven science and engineering: Machine learning, dynamical systems, and control[M]. Cambridge University Press, 2019.

[9] Watkins C. Learning from delayed rewards[J]. 1989.

[10] Mnih V, Kavukcuoglu K, Silver D, et al. Human-level control through deep reinforcement learning[J]. Nature, 2015, 518(7540):529–533.

[11] Sutton R S, McAllester D, Singh S, et al. Policy Gradient Methods for Reinforcement Learning with Function Approximation[C]. In Advances in Neural Information Processing Systems, volume 12. MIT Press, 1999.

[12] Van Hasselt H, Guez A, Silver D. Deep Reinforcement Learning with Double Q-Learning[J]. Proceedings of the AAAI Conference on Artificial Intelligence, 2016, 30(1).

[13] Sanghi N. Introduction to Reinforcement Learning[C]. In: Sanghi N, (eds.). In Deep Reinforcement Learning with Python: With PyTorch, TensorFlow and OpenAI Gym. Berkeley, CA: Apress, 2021: 1–17.

[14] Vinyals O, Babuschkin I, Czarnecki W M, et al. Grandmaster level in StarCraft II using multi-agent reinforcement learning[J]. Nature, 2019, 575(7782):350–354. Publisher: Nature Publishing Group.

[15] Silver D, Huang A, Maddison C J, et al. Mastering the game of Go with deep neural networks and tree search[J]. Nature, 2016, 529(7587):484–489.

[16] Silver D, Schrittwieser J, Simonyan K, et al. Mastering the game of Go without human knowledge[J]. Nature, 2017, 550(7676):354–359.

[17] Silver D, Hubert T, Schrittwieser J, et al. A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play[J]. Science, 2018, 362(6419):1140–1144.

[18] Popova M, Isayev O, Tropsha A. Deep reinforcement learning for de novo drug design[J]. Science Advances, 2018, 4(7):eaap7885.

[19] Gu S, Holly E, Lillicrap T, et al. Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates[C]. In 2017 IEEE International Conference on Robotics and Automation (ICRA), 3389–3396, Singapore, Singapore, May, 2017. IEEE.

[20] Sallab A E, Abdou M, Perot E, et al. Deep Reinforcement Learning framework for Autonomous Driving[J]. Electronic Imaging, 2017, 29:70–76. Publisher: Society for Imaging Science and Technology.

[21] Kaufmann E, Bauersfeld L, Loquercio A, et al. Champion-level drone racing using deep reinforcement learning[J]. Nature, 2023, 620(7976):982–987. Publisher: Nature Publishing Group.

[22] Gazzola M, Hejazialhosseini B, Koumoutsakos P. Reinforcement Learning and Wavelet Adapted Vortex Methods for Simulations of Self-propelled Swimmers[J]. SIAM Journal on Scientific Computing, 2014, 36(3):B622–B639.

[23] Colabrese S, Gustavsson K, Celani A, et al. Flow Navigation by Smart Microswimmers via Reinforcement Learning[J]. Physical Review Letters, 2017, 118(15):158004.

[24] Verma S, Novati G, Koumoutsakos P. Efficient collective swimming by harnessing vortices through deep reinforcement learning[J]. Proceedings of the National Academy of Sciences, 2018, 115(23):5849–5854.

[25] Novati G, Mahadevan L, Koumoutsakos P. Controlled gliding and perching through deep-reinforcement-learning[J]. Physical Review Fluids, 2019, 4(9):093902.

[26] Fan D, Yang L, Wang Z, et al. Reinforcement learning for bluff body active flow control in experiments and simulations[J]. Proceedings of the National Academy of Sciences, 2020, 117(42):26091–26098.

[27] Degrave J, Felici F, Buchli J, et al. Magnetic control of tokamak plasmas through deep reinforcement learning[J]. Nature, 2022, 602(7897):414–419.

[28] Bemporad A, Morari M, Dua V, et al. The explicit linear quadratic regulator for constrained systems[J]. Automatica, 2002, 38(1):3–20.

[29] Fernandez-Camacho E, Bordons-Alba C C. Robust MPC[C]. In: Fernandez-Camacho E, Bordons-Alba C, (eds.). In Model Predictive Control in the Process Industry. London: Springer, 1995: 155–170.

[30] Mainuddin M, Duan Z, Dong Y. Detecting Compromised IoT Devices Using Autoencoders with Sequential Hypothesis Testing[C]. In 2023 IEEE International Conference on Big Data (BigData), 1344–1351, December, 2023.

[31] Han Y, Hao W, Vaidya U. Deep Learning of Koopman Representation for Control[C]. In 2020 59th IEEE Conference on Decision and Control (CDC), 1890–1895, December, 2020. ISSN: 2576-2370.

[32] Shi H, Meng M Q H. Deep Koopman Operator With Control for Nonlinear Systems[J]. IEEE Robotics and Automation Letters, 2022, 7(3):7700–7707. Conference Name: IEEE Robotics and Automation Letters.

[33] Koopman B O, Neumann J V. Dynamical Systems of Continuous Spectra[J]. Proceedings of the National Academy of Sciences, 1932, 18(3):255–263.

[34] 什么是神经网络? - 人工神经网络简介 - AWS[Z]. https://aws.amazon.com/cn/what-is/neural-network/.

[35] 人工神经网络 [Z], March, 2024. https://zh.wikipedia.org/w/index.php?title=%E4%BA%BA%E5%B7%A5%E7%A5%9E%E7%BB%8F%E7%BD%91%E7%BB%9C&oldid=81838422. Page Version ID: 81838422.

[36] Schmid P J. Dynamic mode decomposition of numerical and experimental data[J]. Journal of Fluid Mechanics, 2010, 656:5–28.

[37] Williams M O, Kevrekidis I G, Rowley C W. A Data-Driven Approximation of the Koopman Operator: Extending Dynamic Mode Decomposition[J]. Journal of Nonlinear Science, 2015, 25(6):1307–1346. ArXiv:1408.4408 [math].

[38] Brunton S L, Proctor J L, Kutz J N. Sparse Identification of Nonlinear Dynamics with Control (SINDYc)*[J]. IFAC-PapersOnLine, 2016, 49(18):710–715.

[39] Haarnoja T, Tang H, Abbeel P, et al. Reinforcement learning with deep energy-based policies[C]. In Proceedings of the 34th International Conference on Machine Learning - Volume 70, ICML'17, 1352–1361, Sydney, NSW, Australia, August, 2017. JMLR.org.

[40] Sharma H, Kumar H, Pandey R. Introduction of Reinforcement Learning and Its Application Across Different Domain[J]. International Journal of Scientific Research in Computer Science, Engineering and Information Technology, 1998, 9(6):98–104.

[41] Brunton S L, Brunton B W, Proctor J L, et al. Chaos as an intermittently forced linear system[J]. Nature Communications, 2017, 8(1):19. Publisher: Nature Publishing Group.

[42] Huang S, Dossa R F J, Ye C, et al. Cleanrl: High-quality single-file implementations of deep reinforcement learning algorithms[J]. Journal of Machine Learning Research, 2022, 23(274):1–18.

附　　录

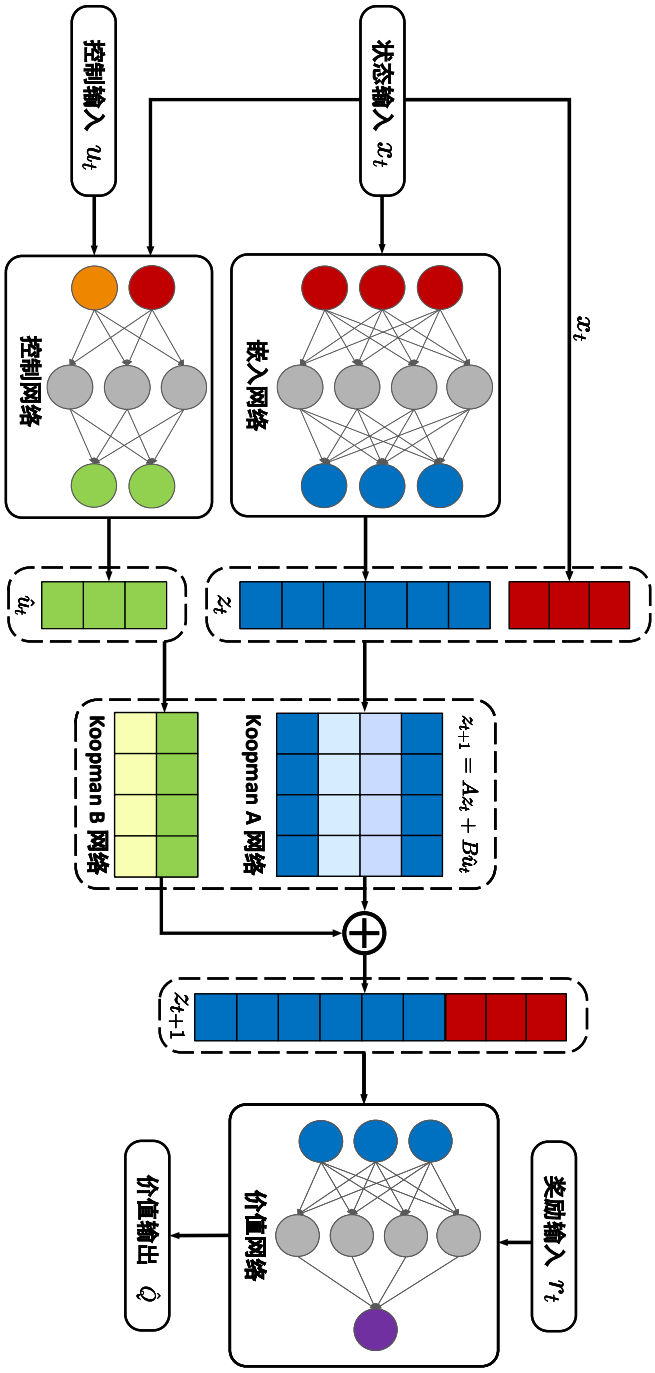

图 A.1　基于非线性形式辅助网络的深度 Koopman 价值网络

致　　谢

首先，原谅我不擅长的用语言和文字表达我的情感，但是我想说，我的感激绝对不是这份致谢可以表达的。

我最最要感谢的是我的母亲，在本科四年的时间里，她总是在我遇到学习生活中难以克服的困难时，给我打下一针“镇定剂”；在我小有成绩时，向自己的朋友们“无限”分享。她是我的母亲，同时也是我最好的朋友，她用言传身教的方式教会了我很多道理：每个人都是一个独立的个体，但是却是群居动物；人和人之间不可能完全理解对方，但是分享和包容能建立起沟通的纽带。所以我在大学四年生活里交到了很多很好的朋友，也学会了怎么学会拥有自己的时间。我一直认为，她是我最亲密的人中，最享受孤独的那个，有了这样的参考，我才能够尽可能积极的生活，谢谢。

我要感谢我的父亲，在学习上，他总是给我保证最好的条件，同时也尊重我的意见，在重要的抉择上，给我最大的自由；犹记得，高考报考志愿，是我独自在房间里，通过网络查阅分数，在三天内自己搞定的。父亲的爱总是无言又默默的，尽管在小事上和我“锱铢必较”，但他总是能够相信我，为我的成长而由衷的高兴，谢谢。

在远离家乡的兰州，有这样一群人，总是在我的身边，帮助我，鼓励我：赵老师，阎老师，实验室的师兄们，涛涛，钊钊，2020 级电信基地班的同学，还有我的三位室友，我们因为兰州而相遇，有些人终因兰州而分别，有些人则将开启和我一起开启在兰大新的生活，分别的话语总是说不完，希望大家都能有美好的未来，谢谢你们!

最后，我要像众多信息学科的学生一样，感谢我的两台笔记本电脑，陪我学习陪我笑，谢谢！我同样也想感谢在过去四年内努力过的自己，一个人总是要在生活中找自己，一个不完美但可（被自己）爱的自己，希望数年后，我能问我自己：“如此简单的梦，有没有实现？”。

毕业论文（设计）成绩表

导师评语

许忞欢同学基于深度 Koopman 算子理论和深度强化学习对非线性系统控制展开了研究。论文详细分析了现有方法的局限性，针对实际复杂系统中控制变量在线性预测中产生畸变等问题，提出了基于深度 Koopman 算子的非线性系统最大熵强化学习算法。相关实验结果表明，提出的结合算法实现了对流行的强化学习的改进。

论文结构合理，层次分明，叙述准确，图表规范，实验设计合理，工作量饱满，完成了开题报告的内容，达到了本科生毕业论文水平。

建议成绩 良 **指导教师（签字）** 赵东东

答辩委员会意见

经答辩小组统一讨论，该论文通过答辩，成绩为 良

答辩委员会负责人（签字） 杨凌

成绩 良 **学院（盖章）**

2024年 5月 29日